Integrity of Advanced Authentication Technologies: Biometrics, Smartcards, and Client Protocols

John R. Michener, Ph.D.
Software Engineer, Consultant
Daniel G. Fritch
Software Engineer, Consultant
Novell Information Security
Novell Inc.

Introduction

A wide variety of advanced authentication technologies and adjuncts have and are being developed to authenticate individuals and enable them to proxy their authentication to and across proprietary networks, intranets and the Internet. Novell is enhancing its authentication architecture to enable it and its customers to use advanced authentication technologies to help authenticate users to their networks. For a large network to successfully use a variety of authentication methods, let alone composites of several methods together, it must be possible to associate appropriate security properties, 'grades', with the methods used and with their environment. In Netware 5, software hooks have been added to Novell's Directory Services (NDS) to allow the Multi Authentication Framework (MAF, which will be submitted to the IETF for general usage) to be integrated with the NDS capability of Graded Authentication. When installed, this will allow the association of appropriate security properties with the implementation and environment of the advanced authentication methods.

The security properties associated with any given authentication implementation are a composite of the technology utilized and critical design and implementation choices. For advanced biometric technologies to fulfil their promise for reliable user authentication, their design and implementation must pay close attention to possible attacks against the integrity of the host system as well as attacks against the implementation of the method and its supporting hardware.

The security properties of an authentication implementation must be viewed in light of where processing is done with respect to the location of trusted computational boundaries (TCB). Methods that perform observations and calculations within "trusted boundaries" (i.e. where an intruder with specified resources and capabilities is unlikely to be able to penetrate) are relatively secure from compromise. Methods that do not perform observations and calculations within "trusted boundaries" are relatively easy to compromise, regardless of the sophistication of the algorithms and protocols that they implement. "Random" nonces and secure time-stamps are critical elements in the provision of resistance to snoop-and-replay attacks. They must be integrated into the system design and implementation where such attacks would otherwise be feasible. Similarly, signatures of authentication information play a critical role in preserving and demonstrating the integrity of this information. This presentation reviews a number of potential vulnerabilities of advanced authentication implementations.

The Threat Environment

Attacks have been increasing in sophistication and potential for damage as the world has become increasingly interconnected and as the value of the activities being conducted by computers and over computer networks has increased. While this is a particularly serious problem for systems that rely upon open computer networks, it is an increasing problem in all business environments. 10 Years ago, passive network snooping attacks were common. At the current time such attacks are ubiquitous and active middleman attacks against systems are well within the realm of competent hackers and their increasingly sophisticated toolkits. As the value proposition develops, we must be prepared for professional system penetrators, who are motivated by money, not by ego. While such attackers will not pass up easier methods if the systems they are attacking are vulnerable to them, we must expect them to use viruses, worms, and custom programmed Trojan-Horse programs to penetrate system defenses and exploit system capabilities.

Let us consider the evolution of authentication methods used by NetWare as an example of the attack/response paradigm that is all too typical of our industry. NetWare 2 originally used a cleartext username / password combination for network login. After this was broken, NetWare V2.15C and the later NetWare 3.XX implemented one-way hashing to prevent network snoopers from accessing user passwords. NetWare 3.X (and bug-fix releases to later NetWare 2 versions) implemented NCP packet signing to provide resistance to hijack and imposture attacks. NetWare 4.XX relies upon NDS authentication, which implements a public-key based zero knowledge proof, that allows the secure delivery of the user public key to the user's client workstation over a hostile wire. This public key is used to generate a session-specific certificate and the key is discarded. NetWare 4.XX supports the NCP packet signing capability as well, although many customers turn this off to increase system throughput. While the NDS authentication is proof against passive and active network snoopers, and if NCP signing is turned on, is resistant to hijack and imposture attacks, the existing implementation IS NOT proof against Trojan Horse attacks. This is particularly true for clients running upon insecure platforms such as Windows 95. Such vulnerabilities are not unique to Novell. Novell NDS authentication and Kerberos implementations rely upon the client to utilize a session-specific certificate or ticket granting ticket, respectively. These are short-term secrets. The X.509 authentication protocols rely upon the client safely storing the long-term secret itself, a far greater vulnerability. The core of the problem is that you can not trust secrets, particularly long-term-secrets, to insecure systems or subsystems without compromising security .

Trust Boundaries in Operating Systems

Computer operating systems and applications are complex and growing more and more complex with each passing year. The interaction of various components of the operating systems and the applications that run upon them defies full understanding if strict guidelines for functional segmentation and operation are not rigorously enforced. Such constraints can allow the behavior of the system to be understood with respect to a specific capability (such as security) to be described. The cost of such a constraint is reduced flexibility — certain capabilities may be limited or proscribed with the result that certain applications my no longer function. There will be increased system overhead as well, as the operating system must verify the acceptability of a operation that an application wants to perform before allowing it. Such oversight includes the questions of: Is the application attempting to read or write in another application's memory space? Is the application attempting to read or write in the kernel's memory space? Is the application attempting to read or write from or to files for which it does not have appropriate permissions? What is the confidentiality level of the user and of the data that is being accessed? Are they compatible? Is the confidentiality level of the output appropriate?

Making a secure operating system, such as Microsoft's Windows NT 3.5 with the SP3, which has been rated as a standalone C2 system when a prescribed set of OS system functionality is loaded, is a difficult proposition. Making a network operating system secure, such as Novell did with its NetWare 4 network security architecture and design, which has been rated as class C2 configuration for network security, is even harder. As in the case with the standalone Microsoft's NT 3.5 C2 product, particular OS configurations are evaluated. Adding additional functionality below the TCB, such as Internet Explorer for Microsoft's NT 3.5 or NFS for Novell's Netware 4.11, break the security model and the C2 evaluated configuration.

For an operating system to have any useable security properties, it must have and enforce a distinction between user (untrusted) space and kernel (trusted system) space and must be able to invoke a trusted path (thereby preventing any snooper program running in user space from intercepting any communications sent across the trusted path) so that the user can authenticate to the kernel. For a general purpose operating system to provide reasonable security properties, the OS enforces memory and disc protection (both in memory and in file access), thereby attempting to restrict the privileges and capabilities of user programs. Special purpose operating systems, such as NetWare, need not rely upon such mechanisms since they do not allow users to run user programs on the system. In general purpose systems, user processes should be protected from one another. The boundary between kernel and user space should utilize processor ring-memory protection hardware. Software functionality should be partitioned between the kernel and the application layer to provide assurance of system functionality and integrity — moving application code into the kernel for increased performance defeats the security and integrity of the architecture. Indeed, the kernel code itself must be carefully designed and layered to minimize undesired side-

effects of the various kernel processes and allow the OS to have consistent and believable behavior. It is the care in designing and verifying the kernel properties and behavior that distinguished high integrity operating systems from those of lower or unknown integrity.

Most of the operating systems used by user's client machines have from negligible to modest security. DOS and Windows (3.X, Win 95, and probably Win 98) have negligible security. Microsoft Windows NT 4 and probably NT 5 can have modest security, but may have negligible security, depending upon what software is installed and how it is configured. The various UNIX implementations (Solaris, HPUX, AIX, ...) typically have modest to moderate security, but they may also have gaping security holes if system applications that break security guidelines, such as NFS or sendmail are installed.

As engineers concerned with issues of system authentication, we can not design (and can rarely even specify) the client operating systems used by our end users. By and large, they will be Microsoft Win 3.X, Win 95, Win 98, NT X.X, with an occasional UNIX system thrown in. The property of the OS is virtually a given (modified by installed applications that may reduce the security of a system). We may have a limited amount of trust in a user system, but the most common situation is one of lack of trust. We need additional sources of trust and we are not going to get it by piling additional layers of software bricks on the quicksand of an untrusted OS/platform.

Trust Boundaries in Authentication SubSystems

Authentication is not identification. The fact that a process or system reported its identity or address does not establish that a binding exists between the identity or address. A process could be misrepresenting its identity and a system with the capability of intercepting network traffic could easily masquerade as a different system. Authentication is the process of confirming a reported identity and (perhaps) establishing a verifiable binding between a process and identity or a system and an address. Without authentication, the identity of systems and processes is suspect.

Authentication subsystems are not general purpose compute engines that are expected to do any task set to them by the system or arbitrary users. Instead, they are special purpose units whose only purpose is to perform certain operations related to security and user authentication very well. The trust that we have in the results reported by these subsystems depends upon how the systems are implemented and the opportunities that the implementation choices present for an attacker to compromise the results. The question is, Where are the trust boundaries?

Trust Boundaries of Token Systems

Password token systems have been in use for well over a decade and are well suited for integration with existing username / password systems. They provide an implementation of "something you have" and have been implemented in a variety of ways: synchronized time-based passwords, challenge-response password tokens; and the ultimate low-tech system, a password list, each of which is used only once. This list is not all-encompassing and other approaches can also be implemented.

The basic characteristic of token systems is that the secret(s) is embedded in the token and shared only with the dedicated password server. The token is trusted and the dedicated password server is trusted. There is no need to share secret information with any other system components and therefore there is no need to trust any other system components. The trust boundaries are the token case and the perimeter of the password server, both of which are typically supplied by the token manufacturer.

Since each password is unique and is used only once, the system is not vulnerable to replay attacks. Since the passwords are dependent upon shared secrets between the token and the password server, the system is not vulnerable to active man-in-the-middle attacks. Unless the token implementation is such that the user enters their password into the token, which then hashes it with one-time password, these systems are vulnerable to two-stage attacks where an attacker snoops the network or installs a keyboard monitoring program to intercept user entered

passwords and then steals and uses the token. The theft resistance of the token system is a security issue of the token manufacturer and is an issue that should be considered by customers.

Trust Boundaries of Smart Card Systems

Smart cards are typically proffered as examples of systems with strong trust boundaries: The trust boundary is the integrity of the die within the smart card. Effective attacks against such a die typically involve the utilization of stroboscopic electron microscopes, ion microprobes, laser microprocessing systems, and the like. These are out-of-band attacks that smart cards are not rated to survive. If an application needs security against such sophisticated attacks, powered, actively watchdogged units with self auto-wipe capabilities, such as found in Tessera PCMCIA units, are indicated. Given the assumptions concerning attacker capabilities, the trust boundaries of a smart card itself are the contact pads on the die.

Having the trust boundaries at the contact pads on the smart card allows the smart card to protect secrets stored in the smart card and it allows the smart card to securely process requested transactions. The smart card can play no role in determining the integrity of the transactions that are submitted to it for processing. It is entirely feasible that one proposed transaction is submitted to the user for approval on the computer screen and an entirely different one is submitted to the smart card for processing or that the user is never informed of a submitted transaction. The vulnerability of the card / user to such attacks is dependent upon the ability of the operating system on the user's machine to maintain full control over the data path to and from the smart card reader and upon the integrity of the application that is using the smart card.

Smart cards do not have keyboards or any integral way of having the user securely enter their identity into the smart card. Many applications that rely upon smart cards also use PIN's entered by the user to validate the user to the smart card, thereby lessening the threat posed by the loss or theft of the smart card. Once the system relies upon it, the secure entry of the smart card's PIN becomes an issue of trust boundaries. The smart card must rely upon some external device to support entry of the PIN and its secure transmission to the smart card. If the smart card reader is a hardened device with its own controller and trusted path from the keyboard to the smart card (which can not be observed by the devices connected to the smart card reader), then the smart card reader establishes a trust boundary. If the smart card reader is a set of electrical contacts that connect the smart card to a computer serial port and all communications with the smart card are via the CPU, with all user input from the keyboard, the smart card system is relying upon the security of the operating system of the user's computer and the integrity of the applications calling the smart card. In such implementations, the smart card PIN is as vulnerable to interception as a user password (which is exactly what it is, since the PIN is a user password to the smart card). In such a situation, an attacker can not know what the secrets are inside the card, since they are hidden behind the smart card's security perimeter, but the attacker probably does not care: The attacker can have the card perform arbitrary actions on behalf of the attacker without the legitimate user's knowledge.

Thus we see that the integration of smart cards into systems can protect certain secret material, but may provide unexpectedly small security benefits. Unfortunately, such vulnerabilities upon host systems and the attendant vulnerabilities to host system compromises can restrict the benefits available from sophisticated authentication systems.

Trust Boundaries of Biometric Systems

Let us consider biometric systems for user authentication based upon imaging some user feature, such as a fingerprint, eye, retinal blood vessel pattern, or facial shape. From the integrity analysis point of view, these are equivalent methods: The user presents himself or herself to the device in some appropriate way. The device images the appropriate feature (and is responsible for detecting forgeries such as latex fingerprints, masks, etc., to whatever detection threshold is deemed appropriate) and may process the resulting image before sending it or the processed resultant to the host computer and eventually the biometric server for user identification / user verification.

We will not deal with the issue of user forgery. This is an issue that is best dealt with by the makers of the authentication systems. Our concern is with system level attacks that can be conducted irrespective of the forgery resistance of the authentication system.

The authentication system has acquired an image that contains features that can be used to identify/authenticate a user. How are these features extracted? Does the authentication system perform this operation itself or does it call upon the host system to perform this operation? The security properties of the two cases are totally different. If the authentication system performs the operation, then it is quite feasible for the feature extraction to be performed within the trusted boundary of the authentication system. If the host system performs the feature extraction for the authentication system, then the authentication system must transfer the image to the host and rely upon it to accurately and correctly extract the features. While it is possible to do this, it is not easy to do so. It requires hardening the communication path between the authentication device and the host, running a secure OS on the host that takes advantage of hardware based ring protection mechanisms, having the feature extraction module integrated into the kernel of the OS, and directly invoking the extraction module so that no user-space software is able to monitor or interfere with the communications from the authentication system to the feature extraction module. If we rely upon a host computer for feature extraction, the integrity of the biometric can be no higher than the integrity of the host computer, which is usually quite bad.

Once we have our features (which is typically a much smaller data block than the original image; but if no feature extraction is done by or for the authentication device, could be the image itself), we must send these features to the biometric server. The biometric server must then respond appropriately, sending its instructions to the appropriate systems. This transmission to and from the biometric server presents a wide range of attack possibilities to the enterprising attacker. The attacker can send a forged feature message to the biometric server. The attacker can record valid feature messages being transmitted to the biometric server and later replay them, substituting the replayed feature message for the feature message that the authentication system attempted to send. The attacker can modify a feature message being sent by the authentication system. Etc.

The essence of the problem is: How is the biometric server to verify that each feature message is authentic, from the authentication system that it claims to be, not an earlier feature message, and is unaltered? Cryptographic techniques are usually used to solve this problem (If the biometric server can actively inject pseudo-random signals into the data acquisition process and extract and interpret these in real time, in which case the biometric server is serving as the feature extractor, the need for cryptographic techniques can be much reduced and may be eliminated.) While an encrypted channel, such as a SSL channel, could be established between the authentication device and the biometric server, doing so is not necessarily appropriate. Such use of encryption technology makes the products and systems subject to technology export controls and may prohibit product usage in some countries that restrict domestic use of encryption technology, such as France and Russia. We need to protect the information from alteration far more than we need to protect it from observation. Having the authentication device create a message digest of the feature message and then sign the digest under their own key (Public or secret, depending upon choice of technology. Public key approaches are more traditional.) allows the server to verify that the message digest came from the authentication device in question and that the feature message has not been altered. Signing does not prevent replay attacks. To prevent replay attacks, the message digest must either be time-stamped before signing (which requires the authentication device to have its own clock and raises the issue of clock synchronization), or the message digest must include a random “nonce”. The nonce is a random number, which for security sake should be issued by the biometric server, which serves to randomize the message digest and prevent replay attacks. Thus the protocol becomes that at the start of user authentication, the authentication device notifies the biometric server that it wishes to perform an authentication and requests a nonce. The biometric server generates the nonce and transmits it to the authentication device (recording the nonce to verify that the correct nonce was used when it receives the signed feature message). The authentication system acquires the user image and performs the appropriate feature extraction. It then takes the nonce, digests the feature message, and signs the resultant, sending the entire structure to the biometric server for recognition and processing. The biometric server takes the structure, verifies the nonce, verifies the signature, and then performs its operations upon the feature message.

Trust Boundaries of Client Authentication Protocols

Three basic approaches to client authentication are widespread in the industry: Storage of username / password pairs, X.509 certificate approaches, and Kerberos ticket approaches.

Username/password storage requires the secure storage of long-term secrets. Username / password pairs stored on a system disc are clearly vulnerable to any routine that can access the appropriate area of the disc if they are stored in the clear, and are vulnerable to Trojan attack programs that compromise the access programs that access this information. In an insecure system, username/password storage is clearly vulnerable, either to direct reads or to Trojan attacks. In "secure" systems where the username/password information is securely stored and the access routines are protected in the kernel, the information is substantially less vulnerable. The integrity of username/password information stored in a smart card is controlled by the integrity issues concerning smart card access and utilization. In all too many cases, these will provide little resistance to an attacker.

X.509 certificate based client approaches require the secure storage of the certificate private key. This is a long term secret and is corresponding valuable. Users tend to assume that X.509 approaches are more modern and inherently more secure that username/password approaches and as a consequence grant them considerably longer lifetimes than passwords. Unfortunately, attacks against the storage of the private key are no more difficult than they are against the storage of the username/password information. Consequentially, in insecure systems, the X.509 private key is easily vulnerable, either directly or indirectly. In "secure" systems, the X.509 private key is much less vulnerable. If the X.509 private key and the associated certificate processing is carried out in a capable smart card, the private key is not vulnerable, even when used with insecure systems. Unfortunately, while the private key may not be vulnerable, the integrity of the host system and the integrity of the card reader / and card usage protocol determines the extent to which an attacker can misuse the X.509 capabilities provided by the card. As presented in the discussion in smart cards, it can be very easy for an attacker to do this.

Kerberos approaches do not involve the local storage of long-term secrets. The workstation logs into a Kerberos "ticket granting server" and as a result of the user authentication to that server, receive a time-stamped "ticket granting ticket", TGT, which can be used to access other network services. The TGT is stored locally in the client software and in insecure systems is vulnerable to Trojan attacks. In secure systems, the TGT is stored below the kernel boundary and is not accessible to attack. The lifetime of a TGT is centrally managed, but is typically less than a day. The integrity of the user password (the long term secret!) used to perform the initial login to acquire the TGT is dependent upon the integrity of the user platform and OS. Kerberos was developed for usage in a community of networked UNIX platforms and relied upon kernel level implementation of the login and TGT managing routines. The integrity properties of Kerberos implemented upon insecure platforms are of course much worse. On insecure platforms Trojan attack programs can easily compromise both the TGT and the user password.

Integrity of Authentication Methods and Technologies

Authentication technologies that have exemplary properties in stand-alone systems can be easily fatally compromised when implemented without proper consideration of the threat environment increasingly typical of network users and internet-connected end-users. The fraction of the commercial user community using C2 (or higher) security workstations interconnected by C2 (or higher) networks is small. In general, the user's workstation is insecure and can not be trusted. Networks are vulnerable to passive snooper attacks and the internet architecture makes it very suitable to active man-in-the-middle attacks. Trojan-horse attacks against operating systems and applications are likely to become ubiquitous in the next decade. Authentication systems dependent upon the host for processing are corresponding at risk from attacks via the host.

In the above discussion we have reviewed issues concerning the application of advanced technologies to authentication. The same issues occur with even greater stakes when attempts are made to secure transactions and processing in an electronic environment built out of individual workstations, general purpose networks, and the Internet. Let us consider a illustrative scenario. If I manage my checking account with a program such as Quicken and print the checks which I will then sign and mail, I have a chance to review the correctness of each transaction

when I sign the check and put it in the billing envelope. Attacks upon the integrity of the program I use can confuse me (a denial of service attack), but they can not result in improper transfers of funds without at least a lapse of my vigilance. The same level of security and oversight is lacking when I perform electronic transactions. A properly implemented attack would either leave no record on my machine that a transaction had occurred or would misrepresent an actual transaction. Only by checking my account information at the bank / trading house would I know what account activity had occurred. The bank / trading house would have no way of distinguishing valid from invalid transactions. All of them would have been properly validated and signed, either by software in the host or by the smart card used for that purpose. The vulnerability occurs because the interface the user utilizes for electronic transactions, the computer display, is in untrusted space and does not have a secure mapping to the transactions that are being conducted. If transactions are to be conducted with a high degree of integrity from untrusted workstations, such as a Win 95 PC, this transaction integrity will have to be supplied by add-in devices that have the integrity that is lacking in the host environment. Just as smart card readers need their own secure PIN entry capability to be useful for authentication in insecure environments, to appropriately handle electronic transactions smart card stations (or equivalent) also need their own transaction display and protocols that require the user to verify and authorize each transaction before the card performs the electronic authorization. The vulnerability discussed here is as present in professional and commercial environments as it is in the home and small business environment. It is a general vulnerability established by our choice to use insecure systems for conducting business that demands high assurance and accountability.

In the last few years we have seen the development of two major attempts to deal with the issue of trusted software in the space of distributed downloaded software modules: Microsoft's signed ActiveX modules and Java's sandbox security model with its associated security manager. Both approaches have merit and both are fatally flawed in implementation. The problem with the implementations available is that both rely upon user space processes to manage security of other user space processes. There is no provision to protect the security processes themselves from an attacker. If we are to rely upon security monitors to maintain process integrity, the security monitors must run within the trusted computing base (that trusted inner portion of the kernel that handles security functionality): The security monitors must run in trusted OS kernel space and must be protected by the hardware ring memory protection mechanisms. The code for the security monitor, as well as the code for the OS kernel, must be stored in restricted access disc partitions so that untrusted operations are unable to access or modify it. If the operating system is unable to provide this level of support for software security monitors, the security monitors serve primarily to provide the illusion of software security, and may increase security risks by causing users to have undue trust in their environment. Running the security monitor in ring 0 adds security, but adds overhead due to the boundary crossing latencies. For Java virtual machines, this overhead can dominate the performance. The development and optimization of JVM's is an area of active research and commercial development. I have been informed by developers in this area that JVM's running in ring 0 on Intel processors can cross the ring boundaries tens of thousands of times per second with potential security overheads of up to 90% to 99%. A secure loader in ring 0 is less intrusive, since the overhead is incurred only when the module is loaded, not while it is being executed.

Integrity of the Communications Channel: Session Hijacking

In the above discussions we have considered the security properties of the authentication method and implementation. Attackers have been assumed to be either active or passive attackers of the authentication protocol, using attacks either at the host machine or upon the communications channel. Unfortunately, in all too many cases it is quite feasible for an attacker who controls the communication channel to ignore the authentication process itself and simply assume a properly authenticated user's identity. The attacker needs to control any intermediate point in the communications channel between the user and the system to which the user is authenticating.

The attacker monitors the user's authentication to the system and the protocols used. Once the user has authenticated to the system, the attacker cuts the communication channel in use by the user and assumes the channel with the user's identity. From the point of the central system, the attacker is the authenticated user. From the point of view of the user, the communications line was dropped (an all too common occurrence anyway) and it

will be necessary to log in again. The attacker may ignore the user's subsequent attempts to log in or may hinder such logins while the attacker is doing whatever they want in the main system. If the attacker wants to hinder the user's subsequent logins, the attacker has a wide choice of methods: injecting "noise" on the line so that additional logins fail, corrupting the user's network packets, dropping the user's network packets, providing busy signals for dialup lines, etc.

The basic problem here is that while the authentication protocol can establish (to whatever degree of certainty is appropriate) that a specified user is actually at the other end of the communications channel, the protocol need not provide any assurance into the future that the user/entity at the either end of the communications channel is actually the user / entity who conducted the authentication (The problem is actually symmetrical. The user is subject to the same uncertainty. Is the server to which the user is connected the same one to which the user authenticated, or is an impostor?).

This problem has a long pedigree, although all too little has been done about it. In general, dealing with this vulnerability requires the use of signing and/or encryption so that an attacker is unable to assume either party's identity (because it does know the signing or encryption keys). IP 4 has support for packet signing, although the key management issue was never solved and a easily available solution for IP 4 does not exist. The packet signing and packet security provisions for IP 6 are more extensive and considerable work is underway to provide the necessary support for general implementation of IP sec, the security and authentication suite of IP 6. Novell long ago recognized the problem and has made NCP packet signatures over its own network protocol, IPX, available for many years. NetWare 5, the newest version of NetWare, provides support for NCP signing over IP networks. Packet signing and packet security measures utilize low level protocols to maintain communications integrity. It is possible to add integrity to communications channels at higher layers as well. The user can (if the user is legally allowed to) establish encrypted channels between the user's machine and the central system. Attackers of the channel do not know the session key used by the encrypted channel and are unable to take it over.

The use of encryption and/or signing technology increases the value of Trojan attack programs. While analysis or brute force attacks against modern ciphers require extreme efforts, the same is rarely true for attacks against the integrity of the systems running the ciphers and signing code. A small Trojan program that reads/intercepts the keys used by the machine and sends them across the wire using a covert channel (as a separate packet, as coded errors within the packet forward error correcting code, etc.) can easily compromise the session security and allows the attacker to impersonate the user or server while the other party is overconfident of the identity or the user or entity that they are communicating with. It would also be straightforward for an attacker to put their channel hijacking code within the user's machine, although the likelihood that the user might notice something irregular is somewhat higher (If the user is using an external modem with a full bank of status lights, the user might notice that transmissions are occurring when they should not be.). While such Trojan attack programs require more work than a simple snoop-the-keyboard TSR, they are certainly feasible.

Classification of the Security Properties of Authentication Methods, Protocols, and Implementations

In the networked computer era we have gone from the old authentication triad of -- something you know, something you have, and something you are — to a pentad of authentication factors:

Authentication Factors

Public, generally available and not tracked or controlled
Client, a software agent is proxy for a user, perhaps using a Kerberos ticket or X.509 certificate
User knowledge, such as the traditional password
User Possession, such as a password token
User Characteristic, such as one or more biometric parameter

The vulnerability of an advanced authentication protocol to transport attack can be rated. A relatively simple classification follows.

Resistance to Attacks Upon the Transport Link

No Resistance, such as cleartext username and password
Resists snoop and replay, such as password token or Diffie-Helleman protocol
Resists Active man-in-middle, use of shared secrets
Resists session hijacking, data signing and/or link encryption
Opaque to all snoopers, link encryption

The first three of these are related to specific types of attacks that an authentication protocol is likely to be subjected to. The last two of these items deal with transport properties that can provide certain desirable properties for the integrity of an authentication method.

The security properties of the workstation can dominate the overall integrity of the authentication process. In general, these properties are fixed and determined by the platform operating system implementation. A simple rating of these characteristics follows:

Platform Security Characteristics — Resistance to Trojan Attacks

Nil	Software TCB	DOS, Win 95, etc.
Moderate	Mixed TCB	Windows NT, UNIX
Strong	Hardened TCB	Trusted workstation with encrypting NIC

Examples of Security Properties of Authentication Methods

1 Win 95 platform, Novell Directory Services login
password is vulnerable to Trojan attack
session is not vulnerable to man-in middle attack on the wire
session is vulnerable to hijacking

2 Win NT platform, Novell Directory Services login
password is not vulnerable to Trojan attack
session is not vulnerable to man-in middle attack on the wire
session is vulnerable to hijacking

3 Win 95 platform, cleartext username/password login
password is vulnerable to Trojan attack
session is vulnerable to man-in middle attack on the wire
session is vulnerable to hijacking

4 Win 95 platform, unsigned, host evaluated biometric login
biometric is vulnerable to Trojan attack
session is vulnerable to man-in middle attack on the wire
session is vulnerable to hijacking

5 Win 95 platform, smart card, keyboard PIN entry
PIN is vulnerable to Trojan attack
session may not be vulnerable to man-in middle attack on the wire
session is vulnerable to hijacking

6 Win 95 platform, smart card, secure PIN entry
PIN is not vulnerable to Trojan attack
session is not vulnerable to man-in middle attack on the wire
session is vulnerable to hijacking

7 Win 95 platform, password token
password is not vulnerable to Trojan attack
session is not vulnerable to man-in middle attack on the wire
session is vulnerable to hijacking

8 Trusted Workstation with encrypting NIC, password
password is not vulnerable to Trojan attack
session is not vulnerable to man-in middle attack on the wire
session is not vulnerable to hijacking

Comparison

1	Win 95 platform, Novell Directory Services login	Y	N	Y
2	Win NT platform, Novell Directory Services login	N	N	Y
3	Win 95 platform, cleartext username/password login	Y	Y	Y
4	Win 95 platform, unsigned, host evaluated biometric login	Y	Y	Y
5	Win 95 platform, smart card, keyboard PIN entry	Y	N	Y
6	Win 95 platform, smart card, secure PIN entry	N	N	Y
7	Win 95 platform, password token	N	N	Y
8	Trusted Workstation with encrypting NIC, password	N	N	N

What is clear from the above examples and their various properties is that employing advanced technology does not necessarily improve security to the extent that one might naively expect. Unfortunately, we are looking at a chain of protection mechanisms and the strength of the overall system is set by the weakest link. Enhancements to the strength of the strongest links may have little effect upon the system security.

The greatest threat to system security is created by the potential of Trojan attack programs to compromise other routines that are critical to proper operation. Fortunately, large scale deployment of Trojan attack programs does not appear to be widespread (Few organizations would realize that they have been penetrated and even fewer would disclose it!). Efforts similar to those currently employed to restrict computer viruses should allow the minimization (but not the elimination!) of the effects of Trojan attack programs in the business environment: routine scans of all executables for correct checksum values, control of what programs and modules are installed on business systems, the use of monitoring programs reporting to centrally (and securely!) administered management security servers, the use of more secure workstation operating systems (such as NT or UNIX rather than Win 95), and the other standard tools of MIS security administrators. While such efforts should allow the minimization of the effects of Trojan attack programs in the business environment, the home user is likely to have far fewer defenses. With the push to move electronic commerce and transactions to the home environment, the value of attacks targeted against home users will increase. Tools to minimize this risk will probably become as widespread and widely used as anti-virus software is today. Despite the use of such tools, it is impossible to guard or detect all possible Trojan attack programs on low assurance platforms!

Use and Management of Multiple Authentication Protocols

Combinations of authentication methods allow the union of the authentication factors and of the resistance that the composite has to attacks upon the transport link. Combinations of authentication methods have no effects upon the vulnerabilities due to the platform security characteristics. Ratings of the signing strength of the protocols used for packet signing on the channel or of the encryption strength (if used) are not ranked. Typically an organization will settle for a small suite of methods and the appropriate method will be selected or hard-coded into the protocol. If a variety of methods are ranked in a particular order by the organization and it is desired to associate particular methods with different security levels, it is not difficult for an Authentication Management Server (AMS) to efficiently manage the appropriate selection of ciphers and signing methods.

Authentication methods and technologies must be implemented, separately or in combination with one another to provide high assurance that the user is correctly authenticated. Novell has developed a framework for the integration and server management and control of multiple authentication methods. Novell has chosen not to patent this framework, the **M**ulti **A**uthentication **F**ramework, MAF, and is submitting it to the IETF as a proposed open standard for the integration of multiple advanced authentication methods. When integrated with an AMS to assign authentication grades, the MAF allows the flexible and extensible management of a wide variety of authentication methods and implementations.

MAF is a client-initiated but server-managed process that allows the AMS to decide which authentication methods are to be invoked, to manage the order in which they are invoked, to dynamically change the subsequent authentication methods based upon results obtained earlier, and to decide the level of integrity/trust that it wants to

associate with the particular authentication. This associated level of integrity/trust may depend upon the methods used, the workstation that the user is working from, the communications channel that is in use, the user's requested role, or any other pertinent factor. Appendix 1 contains the appropriate selection from an early version of Novell's submission to the IETF of the MAF protocol.

> The authentication client contacts the AMS and notifies it that it wants to conduct an authentication session. The client includes with its authentication request a list of up to 127 authentication methods that it supports. Authentication methods are identified by unique unsigned 32 bit integers. The client can support more than 127 methods, but it can only inform the AMS of 127 of them at a time. If the AMS wants more methods, it will ask for them.
>
> The AMS examines the list of methods provided by the client and determines if the ones that it needs are present. If not, it asks for more methods. If suitable methods are not available, it terminates the session (DO close_session). If the methods it wants are present, it notifies the client that it will start the authentication session with a DO method_XXXXXXXX, where XXXXXXXX is the method identifier of the selected method.
>
> The server and the client both then call their respective versions of method_XXXXXXXX. The method modules then perform their characteristic operations with one another and with the user, communication with one another via the MAF kernel function, ResponseTransport(), which allows the two modules a flexible means of transferring data back and forth.
>
> When the modules are done, the client method module reports DONE to the client and the server method module reports DONE(status) to the AMS. The client then reports to the AMS, DONE, WHAT NEXT?
>
> The AMS then can instruct the client to terminate the session (DO close_session), do another method (DO method_XXXXXXXX), or that it has successfully authenticated (DO auth_done).

The basis MAF protocol is a straightforward extension of SOCKS V5, integrated with the AMS system that is required to manage it. How much value can MAF type approaches bring? The answer is: The values brought by MAF, like all other authentication and security tools, is directly dependent upon the architectural design and implementation quality of MAF. The IETF is concerned only with protocols, not with architecture and implementation issues. Improvements in protocol security and integrity are only of importance when the architecture and implementation of the protocol on both sides are properly done. Without an appropriate design and implementation, the most capable protocol is of little value. Indeed, the value of improperly designed or inadequately implemented protocols is probably negative, since they induce in the user and administrator alike an unjustified trust in the integrity of their system. The old military rule is applicable here.

> If you don't have the appropriate security system or don't have time or facilities to use your security system properly, send in-the-clear. You have only compromised the message itself and everybody knows it. Otherwise you may compromise the entire security system, and no one may know it.

Implementations of MAF, or of other authentication or security protocols, provide little beyond misplaced user trust and false confidence if they are not designed to take advantage of trusted system processes and trusted storage. For example, an implementation of MAF that relied upon user-space control programs, transport modules, and methods would provide false confidence and no integrity, regardless of implementation technology , be it Java, signed ActiveX methods, DLL's, or conventional .exe and .com files.

Our implementation of MAF is designed to work in systems that implement hardware-enforced ring memory models and will work in environments that use secure trusted workstation as well as environments that use insecure PC's. Clearly the level of trust and integrity that can be afforded the trusted workstations is far higher

than the insecure PC's, but the architecture is the same. MAF does assume that the AMS is a trusted system and is not easily compromised. In the following figures, the control flow and data flow used by MAF is illustrated. The figures that follow show that MAF is invoked by the Secure Authentication Services software component. SAS is a kernel level security service in NetWare 5, and as such can invoke the kernel level MAF module. Kernel level routines are not allowed to directly invoke user space objects, but they can prepare the stack and return to a routine that did not call them. This is functionally equivalent to calling the user space routine and is shown by the dotted line. The MAF user space module is equivalent to a UNIX daemon, a functionality that is not present in the NetWare server OS.

Summary

The integrity of authentication methods is closely intertwined with the integrity of the systems which use the authentication subsystems. Insecure platforms limit the integrity of authentication and render all transactions and operations conducted upon these platforms as suspect. Indeed, security monitor tools, such as Microsoft's ActiveX technology, or the Java Security Manager and sandbox language model, even assuming a flawless implementation with no security holes, are as compromised as everything else by the insecurity of the underlying platform and operating system. All authentication implementations that rely upon the underlying platforms for critical processing steps are compromised by the insecurity of the underlying platform. Authentication implementations that encapsulate all critical operations within the authentication device can work with insecure workstations without compromise of the authentication integrity. Servers that manage authentication methods are critical authentication system components (compromise of these servers compromises the entire system security and integrity) and must be implemented with appropriate security and administered with appropriate care. Authentication protocols must recognize the potential for sophisticated man in the middle attacks and must make appropriate provision to prevent these attacks from succeeding. The integrity of the operating systems of consumer PC's and the existing security, authentication, and transaction processing subsystems for use with these systems do not provide a secure base for general electronic commerce or any form of legally binding agreements.

Customers are currently requesting means to efficiently combine multiple, independent, authentication methods for user authentication. Novell has submitted the Multi Authentication Framework to the IETF as a proposed open standard for providing this functionality and allowing the efficient management of multiple authentication methods. A basic overview of MAF and of some of the issues concerning implementation of authentication and security tools has been provided.

Appendix 1
MAF Protocol Description

INTERNET-DRAFT J. Michener, D. Fritch, M. Gayman
<draft-ietf-aft-socks-maf-01> Novell, Inc.
Expires 9 February 2000 9 August 1999

Multi-Authentication Framework Method for SOCKS V5

Status of this Memo

Abstract

SOCKS V5 [RFC 1928] provides a means to select one from among a number of authentication methods but does not provide any means for utilizing multiple authentication methods to obtain certain desired authentication properties.

MAF is a client-initiated but server-managed framework. MAF relies on a trusted Authentication Management Server (AMS) to: 1) Select the authentication methods to be invoked, 2) order the execution of methods at the client, as appropriate, and 3) assign integrity grades to the final, composite authentication after all methods that were invoked have completed.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

Please send comments on this document to the aft@socks.nec.com mailing list.

1. Introduction

During an initial SOCKS V5 negotiation, the client and server negotiate an authentication method.

The METHOD value to invoke this proposed multi authentication framework (MAF) SHALL be X'08' (this value falls within the IANA assigned range indicated in RFC 1928 and was assigned by IANA to this proposed method in August 1998.)

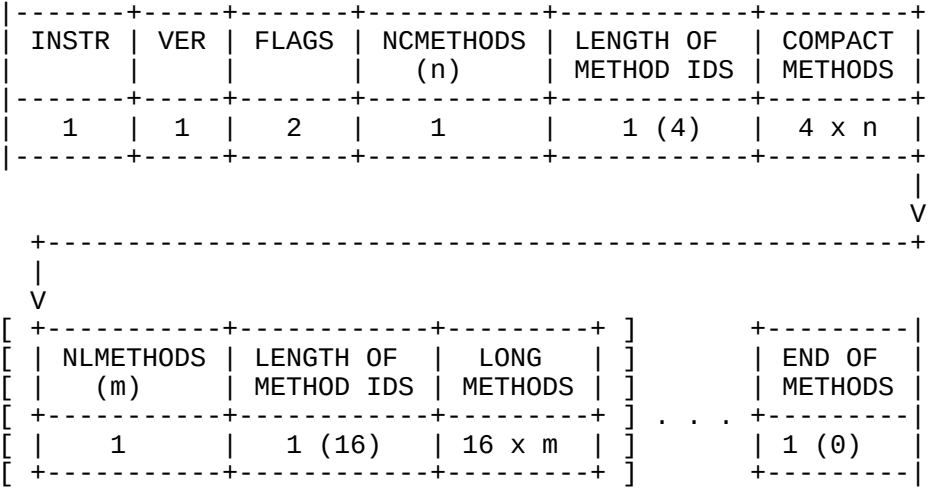
128-bit (16-byte) UUIDs (Universally Unique Identifiers, as defined in [1]) are one form of method identifier used in this protocol. GUIDs (Globally Unique Identifiers), such as those generated by development tools from Microsoft, Incorporated, are suitable as UUIDs.

Unless otherwise specified, all integer values larger than one byte will appear in the protocol messages in most-significant byte first order, i.e., network order.

2. Sub-negotiation

Sub-negotiation, as defined by the SOCKS V5 protocol, begins after the client has selected the MAF method within the SOCKS protocol. All aspects of sub-negotiation are conducted under the control of the server.

The client sends an initial MAF version identifier and potential methods list message to the server:



The INSTR field is a byte that specifies the operation being performed. The values defined at this time, and their colloquial names (in parentheses, if established), and the direction of the message (client to server, server to client, or either) are:

X'FF' Failure and disconnect (Failure), server to client
X'00' Success (Done), server to client
X'01' MAF methods supported (Can Do or List), client to server
X'02' Request additional MAF methods supported (Send More Can Do), server to client
X'03' Do, server to client
X'04' What next, client to server
X'05' Process, either
X'06' Acknowledge, either

X'07' Process OEM specific (OEM), either

To start the sub-negotiation the INSTR field is set to ``MAF methods supported'' (Can Do), X'01'.

The VER field is a byte and is set to the version of the MAF protocol. At this time VER will be X'01'.

The FLAGS field is an unsigned 16-bit value. At this time it is set to X'0000'. It provides for future tuning or extensions of the protocol.

The MAF method identifiers in this second version of the design have two possible formats: A compact or short format consisting of well known 32 bit (4-byte) unsigned integer values and a long format consisting of 128 bit (16-byte) UUIDs. Compact MAF methods IDs are fixed and unalterable after they have been registered (by the IANA or other authority). UUIDs are fixed and unalterable after they have been generated and then employed to identify a particular vendor's method. Consequentially, MAF methods do not have any version identifications per se and many common version incompatibilities are thus avoided. If a method is later found to be inadequate, the revised compact method identifier SHOULD be registered or a new UUID generated by the vendor and the replacement MAF method module SHOULD be released. A bug found in a released method MAY require a new identifier or MAY retain its former UUID or registered identifier, depending on the nature of the bug and the distribution of the method.

NCMETHODS is a byte containing the number of 32-bit compact method IDs in the COMPACT METHODS field that follows the first LENGTH OF METHOD IDS field, which has a fixed value of 4 above. NLMETHODS is the number of 16 byte UUIDs in the LONG METHODS field that follows the second LENGTH OF METHOD IDS field, which has a fixed value of 16 above. A single byte of 0 follows the last long format method ID. The 32-bit compact method IDs are, as stated in the abstract section, in most-significant byte first order and they are not necessarily aligned on 4-byte boundaries in the packet. The byte values in the 16-byte UUIDs are in their standard order as defined by the reference above to the OSF DCE document that describes the algorithm for generating them.

If the client has more MAF method IDs, in either compact or long form, that it can send to the server, the client includes the compact method ID X'00000002' anywhere in the list of 32-bit values to notify the server that more IDs are available.

It is the prerogative of the server whether or not to ask for the additional method IDs by sending to the client a reply with a value of X'02' in the INSTR field (Send More Can Do) with a value of X'01' in VER.

The packet diagrammed above is a specific instance of the general design for this message, which allows the method IDs to be any size from 1 to 255 bytes and allows from 1 to 255 methods of the same length to be grouped together. The three fields bracketed by [and] constitute the general structure that can be repeated as needed in the message, with different values. The last byte of 0 (which would be the number of methods in the next grouping) is always present after the last group of method IDs. To simplify the logic of constructing this packet, all method IDs of the same length SHOULD be sent together in the message. In consideration of future method IDs of lengths other than 4 and 16, it is likely that the length of a method ID will implicitly indicate the

identification space of the value, i.e., an ID that is 4 bytes long is a registered compact format ID, an ID of 16 bytes is a long format UUID, etc. Two IDs of the same value (disregarding more significant bytes of zeroes or encoding differences) will likely not identify the same method if they are of different lengths, e.g., a future one byte ID of 0x09 will not be the same as a four byte ID of 0x00000009.

Nothing SHOULD be imputed or inferred from the order of the method IDs (either compact or long) in the data sent from the client to the server in this packet. Thus it is allowed that the compact method IDs could follow the UUIDs. Either format of IDs could also be absent from the message.

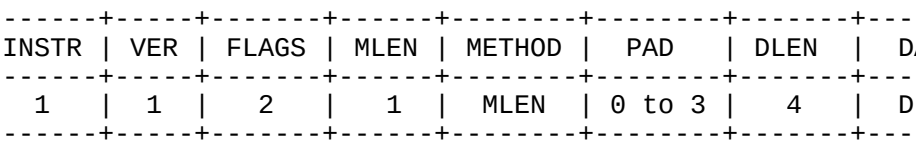
The server MAY select one of the MAF methods identified in either METHODS field (if none of the methods would meet the requirements of authentication policies on the server and the client did not indicate that more method IDs were available, the value of the method selected would be Failure) and send a Do command:

INSTR	VER	FLAGS	MLEN	METHOD
1	1	2	1	MLEN

The INSTR field is set to ``Do'', X'03'. As above, the VER field is set to the version of the MAF protocol. At this time VER is set to X'01' and the FLAGS field is set to X'0000'. The MAF method ID to be performed is entered in the METHOD field. The length of this field in bytes is the value of the single byte MLEN, either 4 for a compact method identifier or 16 for a long identifier.

If the server instructs the client to send more method IDs, via the X'02' ``Request additional MAF methods supported'' instruction, the server will use the FLAGS field to specify either a relative list (the client is to send only methods IDs that have not already been sent) or an absolute list (the client is to start sending the method IDs again as the original list, starting with the first method ID it sent). The FLAGS field will be X'0000' for a relative method ID list or X'0001' for an absolute method ID list.

The client and the server protocol managers then call the appropriate executable modules, or subroutines, to run the specified authentication method. It is anticipated that each method would be a separate binary, executable file. The mapping of method IDs, in either compact or long form, to the names and paths of the files containing the executable code is an implementation issue and is not addressed here. The exchange between the selected client and server modules will use the following data packet:

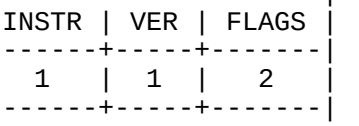


INSTR	VER	FLAGS	MLEN	METHOD	PAD	DLEN	DATA
1	1	2	1	MLEN	0 to 3	4	DLEN

The INSTR field is set to ``Process'', X'05'. As above, the VER field is set to the version of the MAF protocol. At this time VER is set to X'01' and the FLAGS field is set to X'0000'. The ID of the MAF method being performed is present in the METHOD field. The length of this field in bytes is the value of the single byte MLEN, either 4 for a compact method identifier or 16 for a long identifier. The particular data being processed is sent to the client from server or from the server to the client in the DATA field as a byte array, with the length of the array specified in the DLEN field. The PAD field is 0 to 3 bytes of unspecified values to align the DLEN field to a 4-byte boundary relative to the beginning of the packet.

Exchange of data between the method on the client and the method on the server, via ``Process'' packets, continues as long as the two need to run during the particular authentication process.

The client and server methods return success or failure to their respective client and server protocol manager modules. In either outcome case, the client sends the following message to the server:



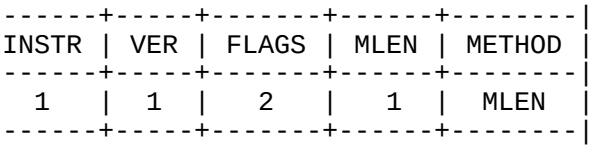
INSTR	VER	FLAGS
1	1	2

The INSTR field is set to ``What next'', X'04'. At this time VER is set to X'01' and the FLAGS field is set to X'0000'.

In the event of failure of an authentication method or of the authentication process, the server MAY instruct the client to close (disconnect) the connection.

If the method succeeded, or if it failed and the server does not need to direct the client to close the connection, the server MAY instruct the client to execute another MAF method module.

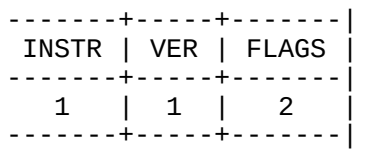
At the end of the process, as determined by policies and controls on the server, the server will send the following in response to ``What next'':



INSTR	VER	FLAGS	MLEN	METHOD
1	1	2	1	MLEN

If the composite authentication process succeeded, the INSTR field will be set to ``Success'', X'00', MLEN to 4, and METHOD to Success, X'00000000'. If the authentication process failed, the INSTR field will be set to ``Failure'', X'FF', MLEN to 4, and METHOD to Failure, X'FFFFFFFF'. In either outcome case, VER is set to X'00' and FLAGS is set to X'0000'.

Upon receipt of either the ``Success'' or ``Failure'' messages by the client, the client will send an acknowledgement to the server. The server will indicate reception of the acknowledge by replying with an acknowledge message as well:



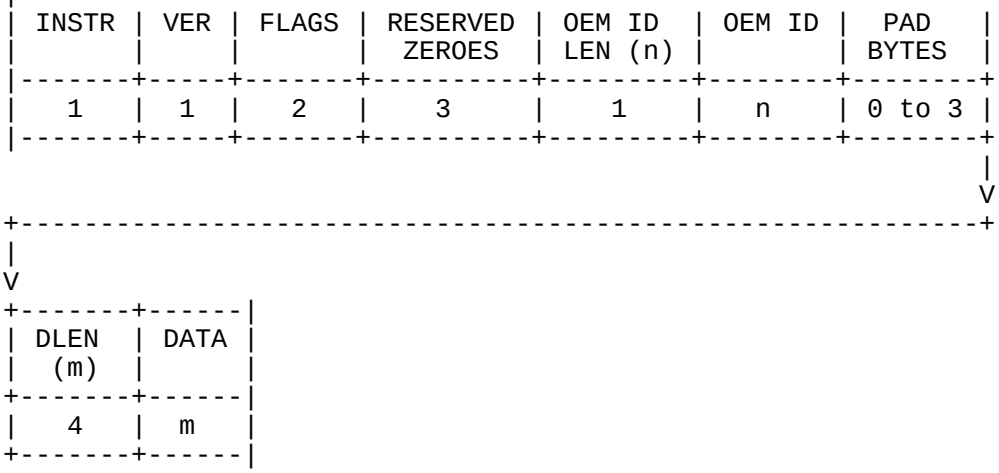
The INSTR field is set to ``Acknowledge'', X'06'. At this time VER is set to X'01' and the FLAGS field is set to X'0000'. The acknowledge message serves to synchronize the MAF protocol client and the server.

3. Process OEM Specific

The ``Process OEM specific'' instruction, X'07', provides a mechanism for the client and server MAF protocol managers to exchange data before or after a method has been invoked (when a given method is running on the client and server, the implementers are free to use the data field of the ``Process'' message to implement any form of communication between the client and the server modules.) The server can send an ``OEM'' message only in response to ``What next'', ``Can Do'', or ``OEM'' messages from the client.

The client can send an ``OEM'' message to the server before sending a ``Can Do'' or ``What next'' message or in response to a previous ``Process OEM specific'', ``Success'', or ``Failure'' message from the server. When one end sends an ``OEM'' message the other end MUST respond with an ``OEM'' message as an acknowledgment. The acknowledgment message can contain no significant data, if desired. When the exchange of ``OEM'' messages is complete, the protocol managers continue with the standard aspects of MAF.

The ``Process OEM specific'' message is composed as follows:



The INSTR field is set to ``OEM'', X'07'. As above, the VER field is set to the version of the MAF protocol. At this time VER is set to X'01' and the FLAGS field is set to X'0000'. The RESERVED ZEROES field is for future use or features. The length in bytes of the OEM ID field is entered into the OEM ID LEN field. In this version of MAF, the length

will be either 4 for compact OEM IDs or 16 for long OEM IDs that are UUIDs. The particular data being processed is sent from one end to the other in the DATA field as a byte array, with the length of the array specified in the DLEN field. The PAD BYTES field is 0 to 3 bytes of unspecified values to align the DLEN field to a 4-byte boundary relative to the beginning of the packet (for the 4- or 16-byte OEM IDs proposed here, this field is absent.) For an acknowledgment message, DLEN can be zero.

If the client or server receives an ``OEM'' message with an OEM ID that it does not recognize or support, it will reply with an ``OEM'' message with the OEM ID set to X'FFFFFFFF' and DLEN set to 0 to so indicate.

3.1 Current Compact OEM IDs

X'FFFFFFFF'	OEM ID not recognized or supported
X'00000000' To X'00000003'	Internal Test IDs
X'00000004'	Reserved
X'00000005' To X'0000FFFF'	Reserved for proprietary IDs, assigned by Novell
X'00010000' and up	General IDs, assigned by the IANA

3.2 Current Compact MAF Method IDs

X'FFFFFFFF'	Failure
X'00000000'	Success
X'00000001'	Internal Test Method ID
X'00000002'	More method IDs are available
X'00000003'	Reserved
X'00000004'	Reserved
X'00000005' To X'0000FFFF'	Reserved for proprietary method IDs, assigned by Novell
X'00010000' and up	General MAF authentication method IDs, assigned by the IANA

4. Security Considerations

MAF enables the efficient combination of multiple authentication mechanisms (allowing the combination of something the user holds, something the user knows, and something the user is). This allows for the reliable establishment of user identity during the authentication session if the methods are appropriately chosen and appropriately managed. The selection of methods and their management are not addressed by MAF. Improper selection of methods and inappropriate management of

the authentication process can invalidate any authentication, including that provided by MAF.

If critical data such as long-term passwords or biometric data are exchanged between the client and the server, appropriate steps SHOULD be taken to secure it at the client (so that an attacker cannot acquire this data), to secure it over the communications channel (using encryption), and to secure this data and its processing at the server. Without appropriate security and integrity at each link in the authentication process, the integrity of the authentication cannot be assured.

5. References

[RFC 1928] Leech, M., Ganis, M., Lee, Y., Kuris, R., Koblas, D., & Jones, L., ``SOCKS Protocol V5'', April 1996.

[1] Steven Miller, ``DEC/HP Network Computing Architecture Remote Procedure Call RunTime Extension Specification Version OSF TX1.0.11'', July 23, 1992.

6. Acknowledgements

We express our thanks to Tolga Acar, Fred Ghiradelli, Tammy Green, and Hal Henderson for assistance in the development of this document.

7. Authors' Addresses

John Michener
Novell, Inc.
122 East 1700 South
Provo Utah, 84606-6194

Phone: +1 801 861-7000
Fax: +1 801 861-2522
Email: jmichener@novell.com

Dan Fritch
Novell, Inc.
122 East 1700 South
Provo Utah, 84606-6194

Phone: +1 801 861-7000
Fax: +1 801 861-2522
Email: dfritch@novell.com

Mark Gayman
Novell, Inc.
122 East 1700 South
Provo Utah, 84606-6194

Phone: +1 801 861-7000
Fax: +1 801 861-2522
Email: mgayman@novell.com

8. Full Copyright Statement